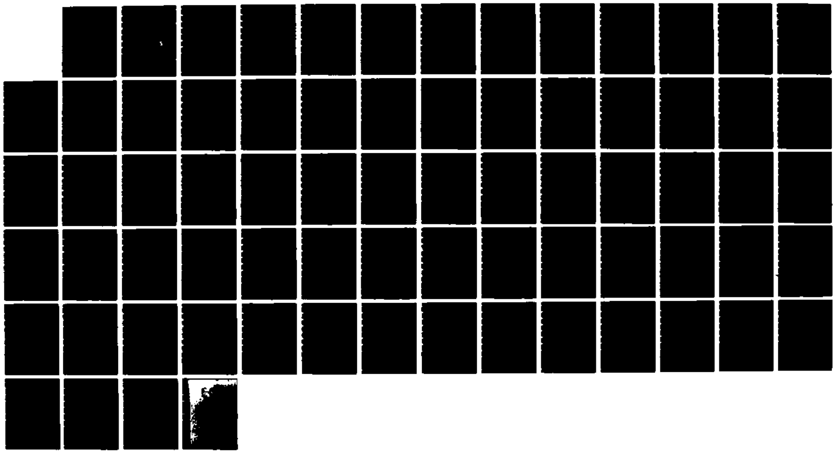
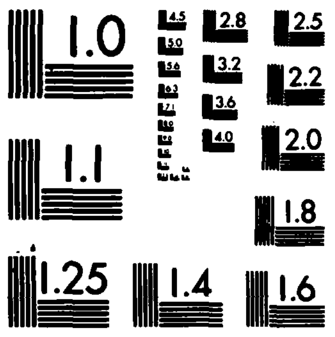



1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR-84-5T ✓	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Installation and Evaluation of SPLICE		5. TYPE OF REPORT & PERIOD COVERED THESIS/~~DISSERTATION~~
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Keith Alan Lewis		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: University of Washington		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433		12. REPORT DATE 1984
		13. NUMBER OF PAGES 63
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASS
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17

Lynn Wolaver

LYNN E. WOLAVER
Dean for Research and
Professional Development
AFIT, Wright-Patterson AFB OH

12 april 84

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

84 04 16 036

Abstract

The project consisted of two parts. The mixed-mode circuit simulation program SPLICE was to be modified and installed on the IBM 4341. Then test circuits were to be simulated to evaluate SPLICE as a simulator. The installation on the IBM consisted of modification of Fortran source code and system functions from the VAX/VMS to the IBM. Separate circuits were simulated to illustrate SPLICE's analysis of logical, electrical, and mixed-mode circuits. SPLICE is not a general purpose circuit simulation program. The available analog elements are limited. SPLICE is of limited use for simulation of LSI-type circuits consisting of logical gates connected by MOSFET switching circuitry.

AFIT/CI/NR 84-5T

AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to ascertain the value and/or contribution of research accomplished by students or faculty of the Air Force Institute of Technology (ATC). It would be greatly appreciated if you would complete the following questionnaire and return it to:

AFIT/NR
Wright-Patterson AFB OH 45433

RESEARCH TITLE: Installation and Evaluation of SPLICE

AUTHOR: Keith Alan Lewis

RESEARCH ASSESSMENT QUESTIONS:

1. Did this research contribute to a current Air Force project?

( ) a. YES ( ) b. NO

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not?

( ) a. YES ( ) b. NO

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency achieved/received by virtue of AFIT performing the research. Can you estimate what this research would have cost if it had been accomplished under contract or if it had been done in-house in terms of manpower and/or dollars?

( ) a. MAN-YEARS ______ ( ) b. $______

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3. above), what is your estimate of its significance?

( ) a. HIGHLY SIGNIFICANT ( ) b. SIGNIFICANT ( ) c. SLIGHTLY SIGNIFICANT ( ) d. OF NO SIGNIFICANCE

5. AFIT welcomes any further comments you may have on the above questions, or any additional details concerning the current application, future potential, or other value of this research. Please use the bottom part of this questionnaire for your statement(s).

NAME GRADE POSITION

ORGANIZATION LOCATION

STATEMENT(s):

Installation and Evaluation of SPLICE

by

Keith Alan Lewis

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science in Electrical Engineering

University of Washington
1984

Approved by [illegible]
(Chairman of Supervisory Committee)

Program Authorized
to Offer Degree Electrical Engineering

Date Feb 27, 1984

Signature Keith Alan Lewis

Date 27 Feb 1984

TABLE OF CONTENTS

LIST OF FIGURES

Abstract

The project consisted of two parts. The mixed-mode circuit simulation program SPLICE was to be modified and installed on the IBM 4341. Then test circuits were to be simulated to evaluate SPLICE as a simulator. The installation on the IBM consisted of modification of Fortran source code and system functions from the VAX/VMS to the IBM. Separate circuits were simulated to illustrate SPLICE's analysis of logical, electrical, and mixed-mode circuits. SPLICE is not a general purpose circuit simulation program. The available analog elements are limited. SPLICE is of limited use for simulation of LSI-type circuits consisting of logical gates connected by MOSFET switching circuitry.

CHAPTER I

Introduction

There has been a steady evolution of general purpose circuit simulation programs since they began to appear in the 1960's. Prior to that time, simulators were often in-house and tailored to specific needs and were limited in the number and types of elements available.

Improved modeling techniques were developed in the early 1960's. The new models were quickly incorporated into the simulators. Hardware improvements had a concurrent evolution and the advancements allowed these simulators to handle larger, more sophisticated and complex circuits.

There have been hundreds of circuit simulator programs developed to date. They can be divided into six basic groups according to the type of analysis used. These analysis types are nodal, state variable, topological (Mason's signal flow), hybrid (Branin state space), ports, and tableau (sparse matrix). Each group has evolved with improvements in simulator performance and capabilities.[1]

Generally the circuit simulators have been either for logic circuits or for analog circuits. Two problems have developed with these separate approaches. First, more and more digital hardware is being used and must be interfaced with a generally analog world.[2] Second, as integrated circuits have increased in size (LSI, VLSI) the ability of simulators to handle them has been exceeded.

A mixed-mode simulation program named SPLICE was developed at the University of California at Berkeley.[3,4] This project was conceived with the objective of installing SPLICE on the IBM 4341 and investigating SPLICE as a mixed-mode simulation program.

The project consisted of two parts. When the project was begun a copy of SPLICE version 1A.2 was available for use on the VAX running VMS. First, this version was to be modified to run on the IBM 4341. The second part of the project was to simulate several circuits and evaluate SPLICE in its performance as a simulator. Midway through the project a new version of SPLICE, version 1.6, became available. It was for the VAX running Unix. It was also evaluated but was not modified for the IBM.

This thesis describes the project in the following way. In Chapter II the modification of SPLICE version 1A.2 from VAX/VMS tape to the IBM 4341 is described. First the procedures used to copy the VAX/VMS tape to the IBM disk are detailed. Then the errors encountered when the program was run and the changes to the Fortran source code to correct these errors are described. Chapter III contains a description of SPLICE, noting what types of elements are available and how an input file is constructed, clarifying and expanding where necessary on the information found in each version's user guide. Both versions of SPLICE are described and the differences noted.

Many test circuits were analyzed using SPLICE. In Chapter IV, the analysis of circuits by SPLICE is described. Three circuits are discussed. The first illustrates the use of SPLICE for analog or electrical analysis. Then an example of logical analysis is presented. Finally, a simple example of mixed-mode analysis is described. Chapter V summarizes the results of the evaluations and discusses the conclusions drawn about SPLICE. Comparisons are made between the two versions of SPLICE as well as using SPLICE on the VAX/VMS, the VAX/UNIX, and the IBM.

Also there are two appendices included in this thesis. Apendix A tells step-by-step how to run a SPLICE file on the IBM 4341 in the University of Washington Electrical Engineering Department. Appendix B describes the step-by-step procedures to run a SPLICE file on the VLSI Consortium VAX/UNIX.

CHAPTER II

Modification of SPLICE Version 1A.2 from VAX/VMS to IBM 4341

When this project was begun SPLICE version 1A.2 was available on a magnetic tape in a form suitable for use on a VAX running VMS. It was desired to run SPLICE on the IBM 4341. Although the procedures described here were used specifically for this copy of SPLICE they outline the steps necessary to accomplish the same translation on any tape in the same format.

There were 12 files on the SPLICE tape. The first four were text files. The first file was the installation procedures, the second file was a program description and user guide, the third file was the initialization file for BLT, and the fourth file was a set of example input files. There were also four Fortran source code files named RATFOR, BLT SPLICE, and SPLOT. Finally there were these same four files in Ratfor source code.

There were two main parts to the modification procedure. First, the files had to be copied onto the IBM disk. Second the Fortran source code programs BLT, SPLICE, and SPLOT had to be modified to be compatable with the IBM system.

2.1 Copy From VAX Tape to IBM Disk

To accomplish the first task the VAX tape had to be copied to the VAX disk. Then a copy was made from VAX disk to CDC tape. This step was necessary because VAX cannot copy form ASCII to EBCDIC. From the CDC tape a copy was made to CDC disk, then to IBM tape, and finally to IBM disk. These steps were accomplished as follows:

1. VAX tape to VAX disk

   a. The tape was given to the operator and it was mounted on the tape drive.

   b. The tape was copied to disk with the following instructions:

```
ALLOCATE MTA0 :
MOUNT MTA0 : SPLICE MT
COPY MTA0 : *.* *.*
DISMOUNT MTA0:
```

These commands caused the files on the tape to be copied to disk. All the files retained the same file names they had on the tape. The file names were listed using the DIR command. This list of file names was needed in the next step.

One problem was encountered in this step. Normal account procedures authorize 500 blocks of disk space. SPLICE is a very large program and it required 3500 blocks of disk space.

2. VAX disk to CDC tape

   a. A blank tape was degaussed and turned into the operator to be mounted on the tape drive.

   b. The files were copied from disk to tape with the following instructions:

```
ALLOCATE MTA0 :
INITIALIZE/DENS=1600 MATAO : SPLICE
MOUNT MTA0 : SPLICE TP
FCOPY filename TP : filename
```

```
ALLOCATE MTA0 :
INITIALIZE/DENS=1600 MATAO : SPLICE
MOUNT MTA0 : SPLICE TP
FCOPY filename TP : filename
```

Where filename was the filename of each of the files (obtained from DIR in the previous step). This last command was repeated 12 times, once for each file.

After all 12 files were copied the command DIR MTAO: was issued and the files on the tape were listed. These could be checked to see that all the files were copied correctly.

3. CDC Tape to CDC Disk:

a. The tape from step 2 was taken to the Computer Center and turned into the tape library. It was labeled VSN=SPLICE.

b. The copy to disk was done using the utility COPYAI. The job was submitted in batch form with the following steps included:

```
DEFINE,DISK=SPLICE1.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=1.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=2.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120, CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=3.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=4.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=5.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=6.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=7.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=8.
```

```
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=9.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=10.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=11.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
LABEL,TAPE,VSN=SPLICE,PO=R,F=S,NT,SI=SPLICE,QN=12.
COPYAI,TAPE,DISK,1,FL=80,MBL=5120,CV=AS.
REWIND,DISK.
RETURN,TAPE.
```

4. CDC disk to IBM tape

   a. A blank tape was degaussed and taken to the Computer Center tape library. It was labeled VSN=SPLIC2.

   b. The copy was made as follows:

```
ATTACH,SPLICE1.
LABEL,TAPE,VSN=SPLIC2,NT,D=1600,F=S,PO=W,LB=KU,CV=EB.
COPYIA,SPLICE1,TAPE,12,FL=80,MBL=3200,CV=EB.
REWIND,DISK.
RETURN,TAPE.
```

Three items should be noted here. In step 4 above note that CV=EB is specified in both the LABEL and COPYIA statements. Also, steps 3 and 4 together require about $30 of CDC time. Finally, the direct access file SPLICE1 created in step 3 is a vey large file. After the copy is completed and verified this file was purged. Failure to do this could lead to expensive storage costs.

5. IBM tape to IBM disk:

   a. The tape was given to the operator and it was mounted on the tape drive and attached.

b. The copy was made using the following EXEC FILE:

```
CDC EXEC A

FI INMOVE TAP1 (RECFM F LRECL 80 BLOCK 3200 DEN
1600 LEAVE
FI OUTMOVE DISK &1 &2 A (RECFM FB LRECL 80
BLOCK 800
MOVEFILE
```

Commands issued were:

```
CDC filename filetype
```

where filename and filetype were for each of the 12 files on the tape. This command caused the next file on the tape to be read onto disk with this filename and filetype.

The 12 files were read as follow:

filename	filetype
1. INSTALL	DOC
2 DOC	DOC
3. SPLIXX	DOC
4. INPUT	DOC
5. RATFOR	FORTRAN
6. BLT	FORTRAN
7. SPLICE	FORTRAN
8. SPLOT	FORTRAN
9. RATFOR	RATFOR
10. BLT	RATFOR
11. SPLICE	RATFOR
12. SPLOT	RATFOR

This complet[illegible] rst main part of the translation procedure. T[illegible] iles were on the 4341 disk.

2.2 Modification of Fortran Source Code

The Fortran source code files BLT, SPLICE, and SPLOT needed to be able to run on the IBM 4341. These three files contained more than 16,000 Fortran statements. Each of these three files was compiled. This generated 270 diagnostics. Many of these were repeated. An example of each type of diagnostic and the way it was cleared follows:

1. USE OF HOLLERITH CONSTANT FOR DATA INITIALIZATION IS NON-STANDARD

   This diagnostic was repeated many times. It produced a warning only, so no correction was made. Later it was found that this type of data initialization was crucial to the running of BLT.

2. INVALID CHARACTER FOUND WHILE CLASSIFYING THE STATEMENT.

   This diagnostic was generated by statements written in lower case. They were rewritten in upper case.

3. USE OF HOLLERITH CONSTANT VALID ONLY IN A FORMAT STATEMENT.

   This error message was from call statements which used hollerith constants as arguments. The hollerith string was simply replaced by character string, i.e., CALLEROR(6hmodels) became CALLEROR('models').

4. FORMAT STATEMENTS CONTAINS AN INVALID FORMAT CODE.

   These errors were from format statements which attempted to specify a single space with the notation ...,X,... . This was changed to ...,1X,.... .

5. STATEMENT AFTER AN ARITHMETIC IF, GOTO, STOP, ASSIGNED GOTO, OR RETURN HAS NO STATEMENT NUMBER. There were two statements with this error message. They were mentioned in the installation procedures, which said to ignore them.
6. The subroutine TIMRB contained many statements which had VAX accounting-type symbols. These all produced an INVALID CHARACTER diagnostics. All of these statements were deleted and this entire subroutine circumvented by placing a RETURN as the only statement in the subroutine.
7. DATA STATEMENT CAN ONLY BE USED TO ENTER DATA INTO A COMMON BLOCK IN A BLOCK DATA SUBPROGRAM. This was a repeated diagnostic. The correction was made by inserting BLOCK DATA SUBPROGRAMS wherever this diagnostic appeared. Only one unnamed BLOCK DATA subprogram may be in a main program. Others were named KEITH1,KEITH2, etc.
8. Open statements caused some of the INVALID CHARACTER DIAGNOSTICS (#2 above). In addition to rewriting them in upper case the carriage control option was deleted.

In addition to the Fortran diagnostics which needed to be cleared, there were other problems encountered. These problems and the solutions follow:

A. The programs BLT, SPLICE, and SPLOT use a function from the VAX called %loc. This function calculates the address of an argument as an integer. It is used in the internal memory management. An equivalent function on the CDC is LOCF. The IBM has no equivalent function.

A recently acquired copy of SPICE, which was rewritten to run on the IBM 4341 had an assembler added to accomplish this function. This assembler, call LOC, was taken from SPICE. It is listed below.

```
LOC          CSECT
*            I = LOC (x)
             SPACE 2
             USING *,15
             L     0,0(0,1)
             N     0,MASK
             SR    15,15
             BR    14
             SPACE 2
             DS    F
MASK         DC    X'OOFFFFFF'
             LTORG
             DROP
             EJECT
             END
```

B. Another VAX function called for in SPLICE is SECOND. This calculates the time of day for use in the program statistics listing. There is no similar function available in IBM Fortran. As with LOC this function was found in the SPICE file as an assembler subroutine. It is listed here.

```
SECOND       CSET
             BC    15,12(15)
```

```
                  DC    X17'
                  DC CL7'SECOND '
                  STM   14,12,12(13)
                  BALR 12,0
                  USING *,12
                  LA    3,AREA
                  DC    X,'8330000C'
                  L     4,16(3)
                  L     2,0(,1)
                  N     4,=X'00FFFFFF'
                  O     4,=X'4E000000'
                  ST    4,16(3)
                  LD    0,16(3)
                  DD    0,=D'10000000.0'
                  STD   0,0(2)
                  LM  14,12,12(13)
                  MVI 12(13),X'FF"
                  BCR 15,14
AREA              DS    4D
                  LTORG
                  DROP
                  EJECT
                  END
```

Both functions, LOC and SECOND, were assembled and TXTLIB functions of both were generated using:

```
TXTLIB GEN LOC LOC
TXTLIB GEN SECOND SECOND
```

C. Another VAX function which was not included in the IBM library functions was JIAND. The 4341 function IAND was substituted. It performs the same task, logical AND.

D. Another problem was encountered when running BLT was attempted. BLT created words from the input file incorrectly. The problem was found in subroutine ENCOD. The statements below encoded characters into words, but they added them together backwards. For example, the letters xyzzy were reconstructed as yzzyx. Two four-letter

```
 IWORD(1)=IAND(CHARS(1),127)+256*(IAND(CHARS(2),127)+256*(IAND(CH
*ARS(3),127)+256*IAND(CHARS(4),127)))
 IWORD(2)=IAND(CHARS(5),127)+256*(IAND(CHARS(6),127)+256*(IAND(CH
*ARS(7),127)+256*IAND(CHARS(8),127)))
```

strings were added together. One string was three blanks and the letter y and the other string was the letters zzyx. Each of the two strings was inverted before adding. The problem was corrected by changing the order of the arguments of the IAND function as shown here. Note the second argument of each IAND function also had to be changed to 255.

```
 IWORD(1)=IAND(CHARS(4),255)+256*(IAND(CHARS(3),255)+256*(IAND(CH
*ARS(2),255)+256*IAND(CHARS(1),255)))
 IWORD(2)=IAND(CHARS(8),255)+256*(IAND(CHARS(7),255)+256*(IAND(CH
*ARS(6),255)+256*IAND(CHARS(5),255)))
```

E. When a run with BLT was attempted it was run with SPLIXX DOC A as the input file. During this initialization process BLT was supposed to create a file BLTINIT and an identical file NEWBLTIN.

BLTINIT was created with no problems, but NEWBLTIN never could be created properly. To compensate for this an identical copy of BLTINIT was made and this extra file was renamed NEWBLTIN. With this done BLT ran correctly.

After all three programs were compiled and the necessary additional functions made available the needed file definitions and procedures to run the programs were developed. This was a trial and error process. The result was the development of the EXEC files listed below. They are named BLT, SPLICE, and SPLOT. By inputing these names as commands the associated EXEC file is run. The necessary filedefs are set and the program is run. The BLT, SPLICE, and SPLOT TEXT files were renamed BLTA, SPLICEA, and SPLOTA. Then each was loaded and a GENMOD module of each created.

The EXEC files are:

BLT:

```
&CONTROL OFF
FI FOR000 DISK BLT ERROR A
FI 5 DISK TEST SPL A
FI 6 DISK BLT OUT A
FI BLTINIT DISK BLTINIT SPLICE A(RECFM VB
FI NWBLTIN DISK NEWBLTIN SPLICE A(REMCF  V
BLTA
FLIST BLT ERROR A
```

SPLICE:

```
&CONTROL OFF
FI 0 DISK SPLICE ERROR A
FI 4 DISK SPLICE STATS A
FI 5 DISK BLT OUT A  (RECFM V
FI 6 DISK SPLICE OUT A
FI 7 DISK SPLICE SCR7  A (RECFM V
SPLICEA
FLIST SPLICE ERROR A
```

SPLOT:

```
&CONTROL OFF
FI * CLEAR
FI 0 DISK SPLOT ERROR A
FI 5 DISK SPLICE OUT A (RECFM V
FI 6 DISK SPLOT OUT A
FI 4 DISK SPLOT SCRTCH A (RECFM V
SPLOTA
FLIST SPLOT OUT A
```

To run the program a SPLICE input file must be created using XEDIT and named TEST SPL A. Then blt is entered as a command, then splice, and then splot. The output plots and listings are in a file named SPLOT OUT A.

When the programs were first run using these steps there were still two problems. The input file would not run correctly if it were in upper case (the user guide said upper or lower case was acceptable). Also the program would produce plots and wplots but would not print. The specific causes of the problems were found by manually stepping through BLT, SPLICE, and SPLOT to determine where the errors were occurring. An additional aid in this process was the corresponding Ratfor source code file of each program. The Ratfor files had comments added, the Fortran files did not. The problems found and the solutions follow.

In BLT, the integerfunctions CAPRP and REP set the symbol type used internally for each character read and for capital (upper case) letters changed them to lower case, which is used internally in BLT. To make this change CAPRP used the following statement:

```
CAPRP=CHAR+32
```

This statement changed the letter, CHAR , to its lower case. To do the same on the IBM the following statement was needed.

```
CAPRP=CHAR-BLANK
```

where BLANK is initialized with:

```
DATABLANK/2h  /
```

The difference is simply the result of different representations for upper and lower case letters in the 128 bit VAX data set and the 256 bit IBM data set.

Similarly the integerfunction REP had to be changed. Using the list of 256 character codes for the IBM the symbol type data table in this integerfunction had to be coded for symbol type:

-1 for symbols
-4 for letters
 0 for numbers

The integerfunction REP used the following statement to enter the table to obtain the correct symbol type:

```
ITEMP=JIAND(ITEM,127)
```

Again, a change was required because the IBM uses a 256 bit character set. The following statements allow the correct entry into the symbol type table, first shifting the argument 24 bits to the right and then performing the IAND function.

```
ITEM1=ISHFT(ITEM,-24)
ITEMP=IAND(ITEM1,255)
```

The problem of not printing outputs was solved simply, but it took a lot of time to find the problem. In SPLOT an integer declaration statement in subroutine PRNT had omitted the variable GETPT.

After these corrections were made the programs ran correctly as follows:

BLT	processes input file	TEST	SPL	A
	creates output file	BLT	OUT	A
SPLICE	processes input file	BLT	OUT	A
	creates output file	SPLICE	OUT	A
SPLOT	processes input file	SPLICE	OUT	A
	creates output file	SPLOT	OUT	A

Also each creates an error file if errors are detected. They are:

```
BLT     ERROR A
SPLICE  ERROR A
SPLOT   ERROR A
```

The file BLT ERROR A, in addition to listing errors detected by BLT, contains a list of the input file. It also contains a node map which numbers the nodes. Nodes are referred to in SPLICE ERROR A by these node numbers rather than the node names from the input file. In addition SPLICE creates a file named SPLICE STATS A which lists run statistics, such as set up time, run time, number of nodes, etc.

CHAPTER III

Features of SPLICE Versions 1A.2 and 1.6

SPLICE is a mixed-mode circuit simulation program developed at the Electronics Research Laboratory, University Of California at Berkeley. [3,4] It consists of three separate programs; BLT, SPLICE, and SPLOT. BLT, the Berkeley Language Translator, is a pre-processor. It processes an input file which contains the description of the circuit and the analysis requests. SPLICE is the simulator. It performs the analyses on the circuit. SPLOT is the post-processor. It processes the output file from SPLICE. SPLOT generates the plots and/or listings requested in the input file.

There are two versions of SPLICE discussed in this thesis. They are version 1A.2 and version 1.6. Following is a description of each of these versions. The differences are great enough to warrant separate discussions. A very detailed description of each version and specific rules about its use can be found in the version User Guide/Reference Manual.

3.1 SPLICE Version 1A.2

This early version of SPLICE performs logic and nonlinear transient timing analyses simultaneously. The program uses nodal analysis techniques. It establishes a node map from the circuit description in an input file.

Nodes are flagged as logic or timing nodes. Fan-in and fan-out tables are generated for each node. Each node is analyzed as the simulator steps through time.

Timing node voltages are approximated at each time step using the Newton-Raphson method, one iteration only. [5] This method requires that each node has a non-zero capacitance to ground.

Logic analysis is also performed as the simulator steps through time. Outputs are calculated using Boolean algebraic operations. The output value for nodes whose inputs have not changed are not recalculated since this output value remains the same.

The user guide mentions a third type of analysis: electrical. It proposed to calculate node voltages using matrix techniques. However, the electrical analysis portion of the program was not operational.

Input File Description

The SPLICE input file contains four types of statements. They are control statements, element statements, model statements, and analysis request statements. An example SPLICE input file is shown in Fig. 3.1.

Control statements are required statements which must be included in every SPLICE input file. There are four of them. The first line in the file must be the single word 'splice'. The second line must be the title statement.

After the analysis request statements a 'go' statement is required. The last statement of the input file must be an 'end' statement.

Element statements describe the circuit topology. Each circuit element is represented by one statement which lists the element name, the nodes to which it is connected, and the model name of the element. For example, the first element statement in Fig 3.1 describes an element named 'a1'. It is connected to nodes '3', '2', and 'a'. The name of the model which describes this element is 'log1'.

```
splice
exclusive-or
;models
model log1 and  : tr=9ns tf=6ns
model log2 or   : tr=9ns tf=6ns
model log3 inv  : tr=9ns tf=6ns
model ain  lsrc : 0 1 0 40u 0u 19u 20u 39u 40u
model bin  lsrc : 0 1 0 20u 0u  9u 10u 19u 20u
;elements
a1   3   2   a   log1
a2   4   1   b   log1
o1   out 3   4   log2
i1   1   a       log3
i2   2   b       log3
ain  a           ain
bin  b           bin
;analysis requests
time 1u 40u
wplot a b out
print a b out
go
end
```

Fig. 3.1 Example SPLICE Input File

Model statements assign a model type and parameters to a model name. The model name is the one referred to in an element statement. The model type is one of the SPLICE built-in model types. The first model statement in Fig 3.1 declares that the model name 'log1' represents a logic AND gate with rise time equal to 9 nanoseconds and fall time equal to 6 nanoseconds.

Note that the first element statement in Fig 3.1, along with the first model statement, describes an AND gate named 'al' with output node '3' and input nodes '2' and 'a', with rise time equal to 9 nanoseconds and fall time equal to 6 nanoseconds. Parameters may be specified or default values may be used. Default values are listed in the user guide.

Analysis request statements are the statements which set the time of the analysis and describe which outputs are to be generated. The 'time' statement sets the total time of the analysis and the time stepsize. The time stepsize is called the MRT, minimum resolvable time.

There are three primary output request statements. Each specifies a type of output and lists the nodes for which the output is requested. A 'print' statement generates a listing of voltage values for timing nodes and logic values for logic nodes. A 'plot' statement generates a plot of the voltages of electrical nodes. The plot is automatically scaled. If the output ranges of different nodes requested by the same 'plot' statement differ greatly, separate scales will be generated. If logic nodes are requested on a 'plot' statement the logic values are printed along the side of the plot. A 'wplot' statement produces a timing diagram plot of the output values. The plots are automatically scaled but all the plots have the same scale.

Although the 'time', 'print', 'plot', and 'wplot' statements are the primary analysis request statements, there are others. They specify options which are available to tailor the analysis as needed. These option statements are listed and described in detail in the user guide.

The electrical node voltages are numerical values. The logic node values can be one of four values. These are 0, 1, X (unknown), and H (high impedance). If the value is falling from 1 to 0 an F is printed. If the value is rising from 0 to 1 an R is printed.

SPLICE Built-in models

The model statements associate model names with model types. These model types are built in to SPLICE and each circuit element must be one of these types.

Logic elements

The logic elements consist of eleven logic gates and logic source. The logic gates are listed in Fig 3.2.

Element	Model Type (used in model statements)
buffer	buf
inverter	inv
and	and
or	or
nand	nand
nor	nor
exclusive-or	xor
exclusive-nor	xnor
sample and hold	sah
multiplexor	mux
tri-state- output gate	tx

Fig. 3.2 Logic Gates Version 1A.2

The buffer passes the input to the output unchanged. The inverter performs the Boolean NOT function. Each buffer and inverter has one input node and one output node. The and, nand, or, nor, xor, and xnor gates perform the associated Boolean algebraic operations. Each of these gates has one output node and up to eight input nodes.

The sample and hold has one input node, one output node, and a control node. With a logic 1 applied to the control the logic value present at the input node is passed to the output node and that value is held until the next 1 is applied to the control node. If the input changes value while the control is on (logic 1) the output also changes to this new value of the input.

The multiplexor has 2 input nodes, one output node, and a control node. The value at the first input node is transferred to the output when the control is logic 0. The value at the second input node is transferred to the output when the control is logic 1.

The tri-state output gate has one input node, one output node, and a control node. The input value is passed to the output when the control is a logic 1. When the control is a logic 0 the output is logic H or high impedance. This type of gate is useful for attaching to a data bus.

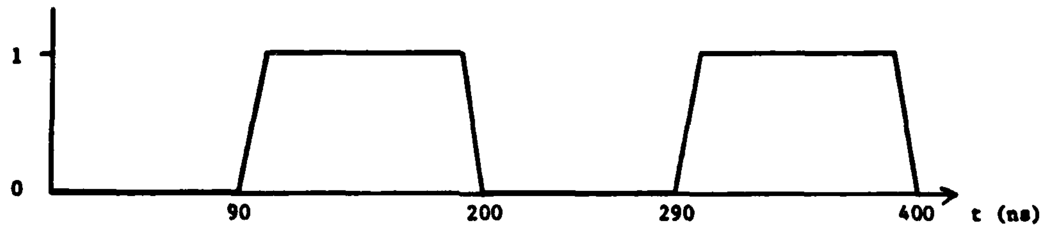
Logic Source

A logic source is defined by a model statement using the following form:

```
model   in   lsrc :  v0  v1  d  p  bp1  bp2 . . .bpn
```

Here, 'in' is the model name, 'lsrc' is the SPLICE model type, 'v0' is the first logic value, 'v1' is the second logic value, 'd' is a time delay, 'p' is the period of the waveform, and 'bp1 bp2 . . . bpn' are up to eleven breakpoints. The source toggles from value 'v0' to value 'v1' at the breakpoints specified and with the period 'p'. The time delay 'd' is applied to all the breakpoints specified in the statement. It should be noted that breakpoints which are not multiples of MRT are ignored.

Fig 3.3 illustrates a logic source. The logic source waveform is shown along with the model statement used to define this source.



```
model   1in   lsrc : 0  1  0  200n  0n  90n  100n  190n  200n
```

Fig 3.3 Logic Source

Timing (Electrical) Elements

The electrical elements available are MOSFETS, grounded capacitors, voltage sources, and voltage rails.

There are six MOSFET models which may be specified. They are listed in Fig 3.4.

MOSFET	Model Type (used in model statements)
n-channel transfer gate	ntxg
p-channel transfer gate	ptxg
n-channel driver	ndriv
p-channel driver	pdriv
n-channel load	nload
p-channel load	pload

Fig. 3.4 MOSFET Models Version 1A.2

Fig. 3.5 Shows the configurations of these models. Each device may be specified as either enhancement mode or depletion mode. This specification is made by declaring the parameter vt (zero-bias threshold voltage) as positive or negative respectively.

Grounded capacitors need to be specified at each electrical node.

Voltage sources are specified exactly the same as logic sources, except v0 and v1 are voltages instead of logic values and the model type of a voltage source is 'tsrc'. There are other electrical elements discussed in the user guide but they are not operational in version 1A.2.

Logical-Electrical Circuit Interface

At nodes where logical and electrical circuit elements are connected together either voltage-to-logic or logic-to-voltage conversion is required. These conversions can be accomplished implicitly using the 'vlevels' and 'llevels' statements or explicitly by specifying discrete interface elements between logical and electrical elements.

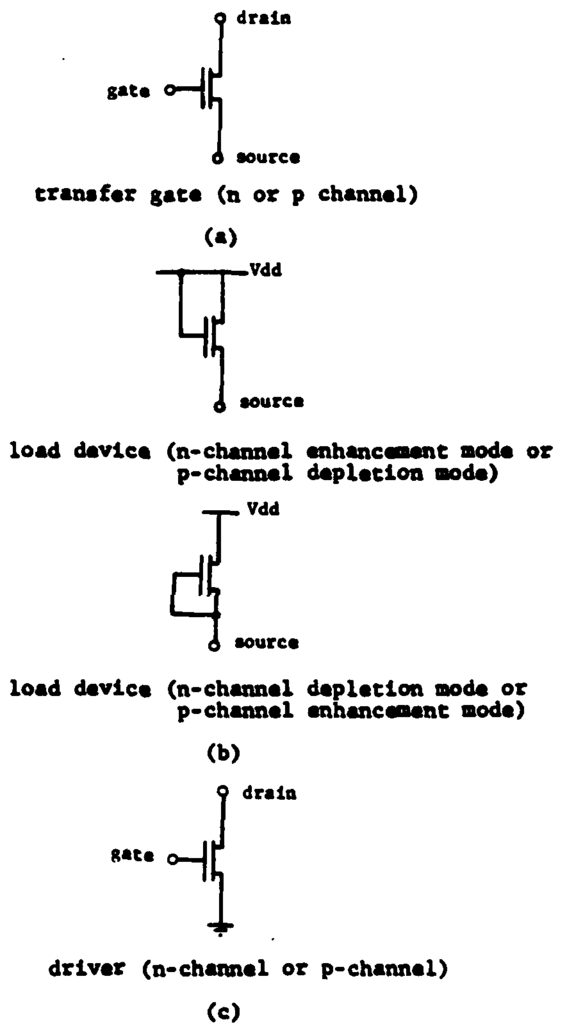

drain
gate
source
transfer gate (n or p channel)
(a)
Vdd
source
load device (n-channel enhancement mode or p-channel depletion mode)
Vdd
source
load device (n-channel depletion mode or p-channel enhancement mode)
(b)
drain
gate
driver (n-channel or p-channel)
(c)

Fig. 3.5 MOSFET Element Configurations
(a) transfer gate (b) load device (c) driver

The 'llevels' statement causes voltage-to-logic conversion to be performed where the output from an electrical element is connected to the input of a logical element.

Similarly, the 'vlevels' statement causes logic-to-voltage conversion to occur where logic output nodes feed electrical input nodes.

There are two discrete interface elements. They are the voltage-to-logic thresholder and the logic-to-voltage converter. An interface element may be specified wherever logical and electrical elements are connected together.

Subcircuits

Subcircuits may also be defined. They are defined using a model statement, the subcircuit file and an ends statement. Fig 3.6 illustrates the use of the subcircuit.

An element statement referring to this subcircuit model is:

```
xl   1   7   9   xorl
```

It refers to the model named 'xorl'. The xorl node 'a' is connected to the circuit node '1', the xorl node 'b' is connected to the circuit node '7', and xorl node 'out' is connected to the circuit node '9'.

```
model xorl subckt : ( a b out )
model andl and    : tr=10ns tf=10ns
model orl  or     : tr=10ns tf=10ns
model invl inv    : tr=10ns tf=10ns
il    1    a        invl
i2    2    b        invl
al    3    2    a   andl
a2    4    1    b   andl
ol    out  3    4   orl
ends  xorl
```

Fig. 3.6 Subcircuit

Fig 3.7 is a summary listing of the main features of version 1A.2.

Summary Version 1A.2

Logic Elements

buffer
inverter
and
nand
or
nor
exclusive-or
exclusive-nor

sample and hold
multiplexor
tr-state gate

logic source

Electrical Elements

nmos transfer gate
pmos transfer gate
nmos driver
pmos driver
nmos load device
pmos load device

grounded capacitor

voltage source
voltage rail

Interface Elements

thresholder (voltage-to-logic)
logic-to-voltage converter

Analysis Options

time (MRT and total analysis time)

plot - single plot
wplot - parallel plots (timing diagram)
print - listing

Fig. 3.7 Summary of Version 1A.2

3.2 SPLICE version 1.6

This version of SPLICE performs timing and logic analysis simultaneously. It operates the same as version 1A.2 except that electrical node voltages are calculated using a new method called iterated timing analysis. This method uses a Newton-Raphson iteration algorithm which is continued to convergence. Logic analysis is accomplished simply by the use of Boolean algebraic operations.

As in the earlier version nodes are still flagged as logic or electrical and fan-in and fan-out tables are generated. Also as in the early version, the latency of portions of the logic circuit is exploited. At a particular timestep in the analysis, if the inputs to a logic gate have not changed the output is left at the same value, not recalculated. The input file description for version 1.6 is the same as for version 1A.2.

Electric node voltages are numerical values. The logic states are different. Version 1.6 uses 9 state logic. This type of logic has the 3 usual states 0 , 1 , and unknown. Additionally there are three strengths associated with each of these states. They are high-impedance, weak, and forcing. This type of logic allows more realistic simulation of logic gates as circuit elements, not just as Boolean operators.

A major difference between version 1.6 and version 1A.2 is found in the built-in models.

Logic elements

There are 16 logic elements. They include 8 Boolean gates and 8 special gates. There is also a logic source. The 8 Boolean gates are listed in Fig. 3.8.

One special gate is the resistive logic gate. The model type is 'res'. This gate passes the signal from input to output with the same logic value, but reduced in strength by one level. Also there is a delay gate which, as its name implies, delays the signal for a specified length of time.

Element	Model Type (used in model statements)
buffer	buffer
inverter	inverter
and	and
or	or
nand	nand
nor	nor
exclusive-or	xor
exclusive-nor	xnor

Fig. 3.8 Boolean Gates Version 1.6

The other six special gates are tri-state type gates. They are available in one or two directional models and can be NMOS, PMOS, or CMOS technology. Each of these gates has an input node and an output node. Unidirectional gates have one control node and bidirectional gates have two control nodes. The gates operate basically the same as before. They pass data in the correct direction for a control which has value logic 1 and output high impedance for a control which has value logic 0.

Logic sources are exactly the same as in the earlier version.

Electrical Elements

The same six MOSFET models found in version 1A.2 are available in version 1.6. Some of the names are different though. Fig. 3.9 lists the MOSFET models.

The configurations of these model types remain the same. Again both NMOS and PMOS devices can be specified as enhancement or depletion mode. There are additional device parameters in this version, the capacitances from each gate to ground.

Element	Model Type (used in model statements)
nmos transfer gate	nmos
pmos transfer gate	pmos
nmos driver	nmosdriv
pmos driver	pmosdriv
nmos load device	nmosload
pmos load device	pmosload

Fig. 3.9 MOSFET Models Version 1.6

Grounded capacitors and voltage sources are the same as in version 1A.2. A grounded capacitor must be specified at each electrical node. There are additional electrical elements available in this version. They are resistors and floating capacitors.

Interface elements are needed between logic and electrical elements. These may be specified implicitly using 'vlevels' or 'llevels' or they may be specified as discrete elements. The discrete interface elements are the voltage-to-logic converter and the logic-to-voltage converter.

The analysis request statements are exactly the same as in the earlier version. Some of the options have changed. The user guide has a detailed description of the options available.

Fig. 3.10 is a summary listing of the main features of version 1.6.

Summary Version 1.6

Logic Elements

buffer
inverter
and
nand
or
nor
exclusive-or
exclusive-nor

transfer gates
- uni- and bidirectional
- nmos, pmos, or cmos

resistive gate
delay gate

logic source

Electrical Elements

nmos transfer gate
pmos transfer gate
nmos driver
pmos driver
nmos load device
pmos load device

resistor
floating capacitor
grounded capacitor

voltage source

Interface Elements

voltage-to-logic converter
logic-to-voltage converter

Analysis Options

time (MRT and total analysis time

plot - single plot
wplot - parallel plots (timing diagram)
print - listing

Fig. 3.10 Summary of version 1.6

Two versions of SPLICE were available. First, version 1A.2 was available for the VAX running VMS. This version was modified to run on the IBM 4341 as described in Chapter II. Second, version 1.6 was available for the VAX running UNIX.

Although version 1.6 has more elements available and improved analysis algorithms as described above, both versions are limited in analysis performance, especially for electrical (analog) circuits. The only type of analysis available is a transient analysis beginning at time zero.

The analog circuit elements available in SPLICE are limited. They do not include bipolar transistors, induct-tors, diodes, and others. However, the program was not intended to be used as a general purpose simulator. The introduction to the user guide states that SPLICE was speci-fically designed for use in the simulation of LSI-type cir-cuits basically using logic analysis on large, repeated segments of the circuit and connecting these segments with MOSFET switching circuits.

CHAPTER IV

SPLICE Analysis

In this chapter separate examples are presented to illustrate the use of SPLICE for analog or electrical analysis, logical analysis, and mixed-mode analysis.

4.1 Electrical Analysis

The electrical, or timing, analysis performed by SPLICE can best be illustrated using an example. Consider the NMOS inverter circuit shown in Fig. 4.1. [6, 7]

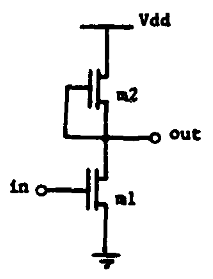


Fig. 4.1 NMOS Inverter

The SPLICE input file describing this circuit is shown in Fig. 4.2. The construction of SPLICE inputs files is explained in Chapter 3. The file includes requests for a combined plot and a parallel plot of input and output voltages.

Two circuit characteristics of interest in this inverter are the output voltage when Vin = 0v and the output voltage when Vin = 5v. The other circuit characterisics of interest are the rise time and the fall time of the inverter. To determine these circuit characteristics the following MOSFET characteristics were used:

Vds	drain-source voltage
Vgs	gate-source voltage
vt	zero-bias threshold voltage
wol	aspect ratio (channel width divided by channel length)
kp	transconductance parameter
Ids	drain-source current

The MOSFET is assumed to operate in one of three regions, off, non-saturated, or saturated. When Vgs < vt the device is off and Ids $\approx$ 0. When the device is not saturated Vds < Vgs - vt and

$$\text{Ids} = \beta\,[(\text{Vgs} - \text{vt})\text{Vds} - \tfrac{1}{2}\text{Vds}^2] \qquad (4.1)$$

where β = (wol) (kp). When the device is saturated, Vds > Vgs - vt and

$$\text{Ids} = \frac{\beta}{2}\,(\text{Vgs} - \text{vt})^2 \qquad (4.2)$$

The device parameters specified for the driver (ml) were: aspect ratio (wol) = 2.0, zero-bias threshold voltage (vt)= +1.0v, and the transconductance parameter (kp) = $30\text{x}10^{-6}$ A/V^2. For the other device parameters SPLICE default values were used. The same parameters were

```
splice
NMOS inverter - depletion mode load device
;
;models
;
model mos1 nload : wol=0.5 vt=-4.0 kp=25u
model mos2 ndriv : wol=2.0 vt=1.0  kp=30u
model vin  tsrc  : 0  5  0  200ns 0ns 99ns 100ns 199ns 200ns
model v22  tsrc  : 5  5  0  1s  0n  1s
model gc1  gcapr : 5pf
model gc2  gcapr : .1pf
;
;elements
;
m1   out    in   mos2
m2   out         mos1
v1   v22         v22
v2   in          vin
c1   in          gc1
c2   out         gc2
c3   v22         gc1
;
;analysis requests
;
time  1ns 400ns
wplot  in  out
print  in  out
;
go
;
end
```

Fig. 4.2 SPLICE Input File - NMOS inverter

specified for the load device (m2). The values were wol= 0.5, vt= -4.0v, and kp= 25×10^{-6} A/V^2. Again the other parameters used SPLICE default values. Input values are Vin(0) =0v and Vin(1)= 5v.

The output characteristics were calculated using design-type estimation techniques assuming operation of the inverter in the following way: When Vin = 0, the driver is off and the output capacitance charges up from Vout(0) to Vout(1) through the load device, which is not saturated. Here the load is assumed to be capacitive only and lumped in value. When Vin = 5v the device is not saturated, the load device is saturated and the output capacitance, C_L, discharges through the driver. [6, 7]

With the driver off and the load not saturated:

$$Vout(1) = Vdd - Vdsl \qquad (4.3)$$

Vdsl is the drain-source voltage of the load device. Here Vdsl is approximately zero, so:

$$Vout(1) \approx 5.0\ V \qquad (4.4)$$

With the driver not saturated and the load in its saturated region the expression for the load current is:

$$Ids = \frac{\beta_L}{2}\, vt_L{}^2 \qquad (4.5)$$

where the subscript L denotes load device parameters. Driver parameters are denoted by the subscript D. The driver current is from (4.1)

$$Ids = \beta_D\,[(Vgs - vt_D)\ Vds - \tfrac{1}{2}Vds^2] \qquad (4.6)$$

Equating the two currents

$$\frac{\beta_L}{2} vt_L{}^2 = \beta_D[Vgs - vt_D)\ Vds - \tfrac{1}{2}Vds^2] \qquad (4.7)$$

where Vgs = Vin(1) and Vds = Vout(0). Assuming Vout(0) << Vin(1)-vt_D

$$\frac{\beta_L}{2} vt_L{}^2 \approx \beta_D[(Vgs - vt_D)\ Vout(0)] \qquad (4.8)$$

rearranging terms and solving for Vout(0)

$$Vout(0) = \frac{\frac{\beta_L}{2}\quad vt_L{}^2}{\beta_D\ (Vin(1) - vt_D)} \qquad (4.9)$$

substituting values from the input file:

$$\mathrm{Vout}(0) = .417 \text{ v} \tag{4.10}$$

The rise time was calculated assuming that C_L charges from vout(0) to Vout(1) through the load device with a constant current, I_L .[7] This charging current is:

$$I_L = C_L \frac{\Delta v}{\Delta t} \tag{4.11}$$

where	C	is the output capacitance
	Δv	is Vout(1) - Vout(0)
	Δt	is the rise time, tr

I_L is the saturation current in the load device.

$$I_L = \frac{\beta_L}{2} vt_L{}^2 \tag{4.12}$$

Substituting (4.12) into (4.11) and solving for tr:

$$tr = \frac{C_L\,[\mathrm{Vout}(1) - \mathrm{Vout}(0)]}{\frac{\beta_L}{2}(vt_L{}^2)} \tag{4.13}$$

$$tr = 4.6 \text{ ns} \tag{4.14}$$

The estimate of the fall time assumes that C_L discharges through the driver only. The fall time is divided into 2 parts. First Vout drops from Vout(1) to Vout(1) - vt_D wth the driver still saturated and having constant current I_{D1}.

$$I_{D1} = \frac{\beta_D}{2} (Vgs - vt_D)^2 \qquad (4.15)$$

Here, Vgs = Vin(1) = 5v. To find tfl, the fall time from Vout(1) to Vout(1) - vt_D with constant I_{D1}, where

$$I_{D1} = C_L \frac{\Delta v}{\Delta t} \qquad (4.16)$$

$$\Delta v = Vout(1) - Vout(1) - vtdevice = vt_D$$

$$\Delta t = tfl$$

substituting values and rearranging terms:

$$tfl = \frac{C_L \; (vt_D)}{\frac{\beta_D}{2} [Vin(1) - vt_D]^2} \qquad (4.17)$$

$$tfl = .2ns \qquad (4.18)$$

To estimate the second part of the fall time, C_L is assumed to discharge from Vout(1) - vt_D to Vout(0) through the driver with a non-constant current, I_{D2} where:

$$I_{D2} = C_L \frac{dVout}{dt} \qquad (4.19)$$

where, from (4.1):

$$I_{D2} = \beta_D [(Vin(1) - vt_D) Vout - \tfrac{1}{2} Vout^2] \qquad (4.20)$$

In (4.20) Vout(1)= 5v and Vout varies from Vout(1) - vt_D to Vout(0). Equating (4.19) and (4.20):

$$C_L \frac{dVout}{dt} = D [(Vin(1) - vt_D) Vout - \tfrac{1}{2} Vout^2] \qquad (4.21)$$

Solving for Vout:

$$\mathrm{Vout} = [\mathrm{Vout}(1) - vt_D] \frac{2e^{-t/\tau_D}}{1 + e^{-t/\tau_D}} \qquad (4.22)$$

where

$$\tau_D = \frac{C_L}{\beta_D [\mathrm{Vout}(1) - vt_D]} \qquad (4.23)$$

Then tf2 was found by letting Vout equal Vout(1) - vt_D and solving (4.22) for t, then solving (4.22) for t with Vout equal Vout(0). Finally, tf2 is estimated as the difference of these two times. The result is:

$$tf2 = 1.2\mathrm{ns} \qquad (4.24)$$

now to find tf:

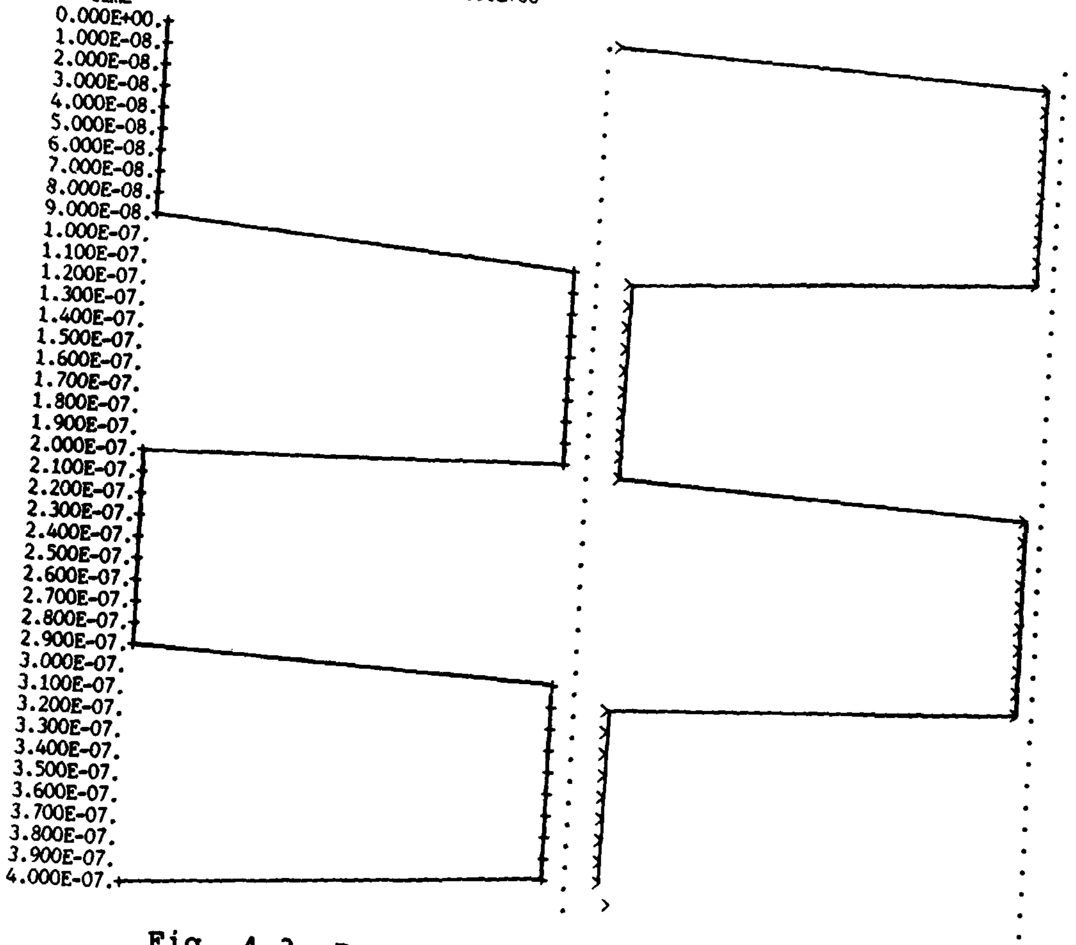
$$tf = tf1 + tf2 = 1.4\ \mathrm{ns} \qquad (4.25)$$

The SPLICE input file shown in Fig. 4.2 was run using both versions of SPLICE. The parallel plot of Vin and Vout is shown in Fig. 4.3. The circuit was also run using SPICE. The circuit characteristic values are tabulated in Fig. 4.4.

Note the model statement for V22 in the input file in Fig. 4.2. At first this file was run with this drain voltage source specified implicitly as a parameter in the load device model statement. Each attempt to run the program produced a system error message indicating

```
        **** splice version 1A.2 ****
nmos inverter - depletion mode load device;


legend:
+ <=> in        :elect      :
> <=> out       :elect      :
 scale: minimum= 0.000E+00 maximum= 5.000E+00
```



```
  time
0.000E+00.
1.000E-08.
2.000E-08.
3.000E-08.
4.000E-08.
5.000E-08.
6.000E-08.
7.000E-08.
8.000E-08.
9.000E-08.
1.000E-07.
1.100E-07.
1.200E-07.
1.300E-07.
1.400E-07.
1.500E-07.
1.600E-07.
1.700E-07.
1.800E-07.
1.900E-07.
2.000E-07.
2.100E-07.
2.200E-07.
2.300E-07.
2.400E-07.
2.500E-07.
2.600E-07.
2.700E-07.
2.800E-07.
2.900E-07.
3.000E-07.
3.100E-07.
3.200E-07.
3.300E-07.
3.400E-07.
3.500E-07.
3.600E-07.
3.700E-07.
3.800E-07.
3.900E-07.
4.000E-07.
```

Fig. 4.3 Parallel Plot - NMOS Inverter

NMOS Inverter

	estimate	SPLICE 1A.2	SPLICE 1.6	SPICE
Vout(0)	.417v	.325v	.441v	.441v
Vout(1)	5.0v	4.916v	4.997v	5.0v
tr	4.6ns	6ns	7ns	7ns
tf	1.4ns	1ns	3ns	2ns

Fig. 4.4. NMOS Inverter Circuit Results

an underflow problem. Next Vdd was specified explicitly as a vrail, with the same results. Then I noticed in the user guide example a voltage rail specified as a source with a 1 second period and both voltage levels equal to 5 volts, i.e. a rail for 400ns analysis. The file was run with the rail specified in this manner and no underflow problems were encountered.

4.2 Logical Analysis

The logic analysis portion of SPLICE was intended as means to simulate macro sections of LSI-type circuits. A section of an LSI circuit is assumed to be repeated often in the overall layout. A detailed electrical analysis can be performed on the section to be used in the macro. Based on the results of the electrical analysis the circuits can be modeled using logic gates. With the appropriate parameters assigned to the logic circuit components the macro can adequately simulate the repeated section.

The logic analysis technique used in SPLICE is quite simple. The nodes are checked at each timestep to see if any of the inputs to a node has changed. If the inputs to a node are unchanged from the last timestep there is no need to recalculate the output from the node. Its value remains the same. If an input has changed the output is recalculated and if it needs to be changed it is scheduled to be changed at the completion of either a rise time or fall time delay. This delay portion of the algorithm allows for realistic tracing of signals that must pass through multiple logic stages.

Note that since the output change is delayed until the elapse of the rise or fall time there is the possibility that the input which caused the output change may itself change back to its original state during this delay. This is the circuit behavior which propogates glitches or spikes. If this occurs SPLICE will not generate the glitch. The program will go ahead and change the output value as scheduled. Although the glitch is not propogated through the rest of the circuit, if the 'gltchflg' option is on the attempted glitch will be reported on the output.

Some of the features of SPLICE as a logic circuit simulator are best illustrated with an example. Consider the toggle flip-flop logic circuit in Fig. 4.5. [8] The SPLICE file describing this circuit is listed in Fig. 4.6.

This circuit operates as follows: The initial states are set by the set and reset signals. The value of q is set to logic zero and qb is set to logic 1. The initial state of nodes 1, 2, 5, and 7 is zero and the initial state of nodes 3 and 6 is one.

If t is zero, q and qb maintain their values regardless of the value of the clock signal, cl. However, if t is equal to one this is not the case. From the initial state above with t equal to one when cl changes to

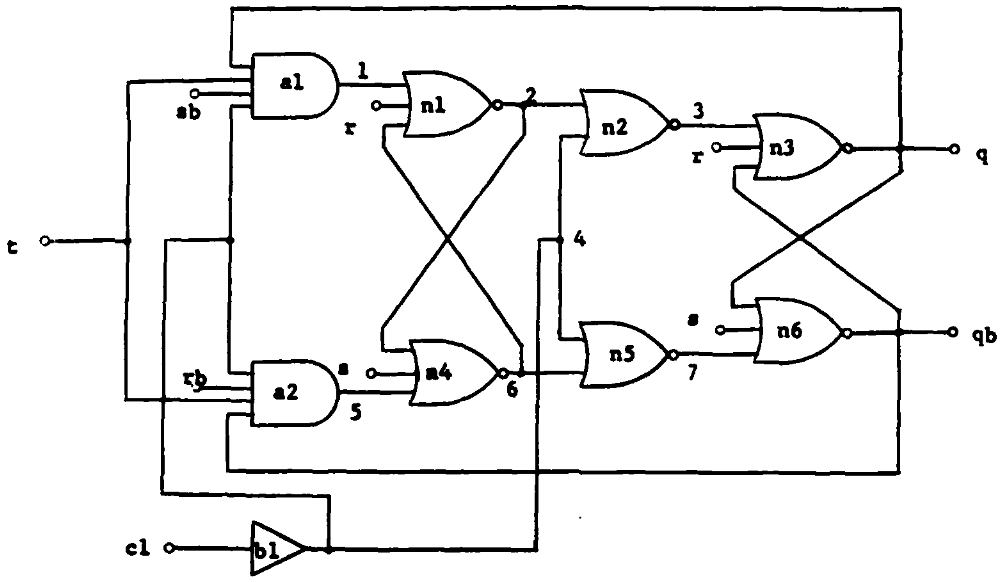


Fig. 4.5 Toggle Flip-flop Circuit

one node 5 changes value from zero to one. Node 1 remains equal to zero. Next, node 6 changes from one to zero. This causes node 2 to change value from zero to one.

When cl had first changed from zero to one, it had caused both nodes 3 and 7 to have value zero, but q and qb had not changed because of the feedback to the inputs of n3 and n6. Now however with nodes 2 and 6 at their new values when cl drops from one back to zero node 7 changes value to one. This causes qb to change to zero which is fed to n3, which then causes the output value of q to become 1. This single cycle of operation of the circuit is shown in a timing diagram in Fig. 4.7.

```
splice
T flip-flop
;
;models
;
model tin lsrc : 1 0 0 400n 0n 190n 200n 390n 400n
model res lsrc : 1 0 0 400n 0n  10n  20n 400n
model set lsrc : 0 0 0 400n 0n 400n
model clk lsrc : 0 1 0 100n 0n  39n  40n  60n  61n  100n
model and and  : tr=5ns tf=3ns
model nor nor  : tr=5ns tf=3ns
model inv inv  : tr=5ns tf=3ns
;
;elements
;
a1   1    q   t   sb   4   and
a2   5    qb  t   rb   4   and
b1   4    cl               buf
n1   2    1   r   6        nor
n2   3    2   4            nor
n3   q    3   r   qb       nor
n4   6    2   s   5        nor
n5   7    4   6            nor
n6   qb   q   s   7        nor
i1   sb   s                inv
i2   rb   r                inv
tin  t                     tin
clk  cl                    clk
res  r                     res
set  s                     set
;
;analysis requests
;
time  1ns 400ns
pstep 10
wplot  t  q  cl
print  t  q  cl
;
go
;
end
```

Fig. 4.6 SPLICE Input File; T Flip-flop

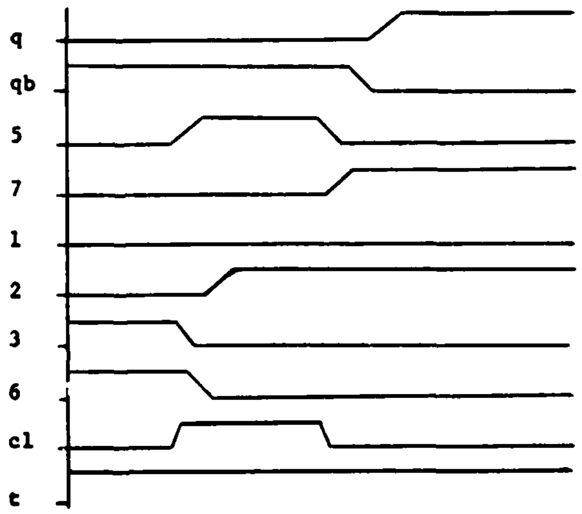


Fig. 4.7 Toggle Flip-Flop Timing Diagram

The overall operation can be summarized as follows: When t is zero, cl has no effect on q and qb. When t is one, q and qb will toggle in value on the trailing edge of the clock signal.

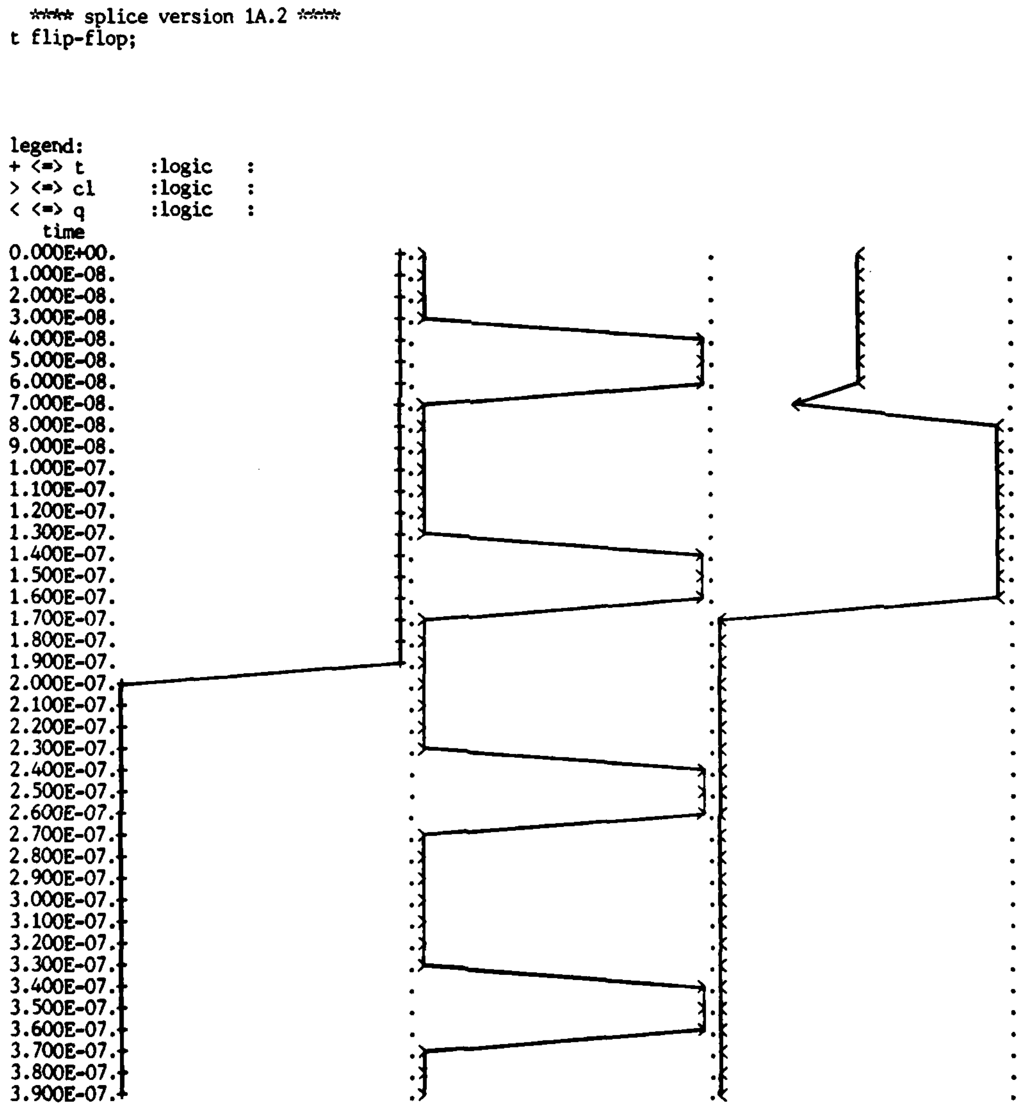
The timing can be checked in the following way. When t is one, after the trailing edge of cl, qb changes value after a rise time delay at n5 and a fall time delay at n6. Then q changes value one rise time delay later at n3.

The parameter values in the example are tr=5ns and tf=3ns. With these values qb should change from one to zero 8 nanoseconds after the trailing edge of cl and q should change from zero to one 5 nanoseconds after qb has reached its new value. The file was run from 0 to 100ns and these times were verified.

When the file was run using version 1A.2 an error was found in the initial state of q. It remained at value unknown until the first clock cycle. It should have been zero after the reset pulse. Fig. 4.8 is the wplot listing from this file using version 1A.2. It verifies the analysis above. When t is one the output q toggles on the clock, when t is zero nothing changes regardless of the clock. Note the error in q until the first clock pulse.

4.3 Mixed-mode Analysis

To analyze a mixed-mode circuit, SPLICE simply divides the logical and analog circuit segments and analyzes them separately and concurrently, interfacing them when necessary. The use of SPLICE to analyze mixed-mode circuits can be illustrated using an example. The static register circuit in Fig. 4.9 is very simple, yet it includes the interfacing necessary in mixed-mode analysis in SPLICE. [9]



```
  **** splice version 1A.2 ****
t flip-flop;


legend:
+ <=> t       :logic   :
> <=> cl      :logic   :
< <=> q       :logic   :
   time
0.000E+00.
1.000E-08.
2.000E-08.
3.000E-08.
4.000E-08.
5.000E-08.
6.000E-08.
7.000E-08.
8.000E-08.
9.000E-08.
1.000E-07.
1.100E-07.
1.200E-07.
1.300E-07.
1.400E-07.
1.500E-07.
1.600E-07.
1.700E-07.
1.800E-07.
1.900E-07.
2.000E-07.
2.100E-07.
2.200E-07.
2.300E-07.
2.400E-07.
2.500E-07.
2.600E-07.
2.700E-07.
2.800E-07.
2.900E-07.
3.000E-07.
3.100E-07.
3.200E-07.
3.300E-07.
3.400E-07.
3.500E-07.
3.600E-07.
3.700E-07.
3.800E-07.
3.900E-07.
```

Fig. 4.8 Parallel Plot - T Flip-flop

To analyze this circuit using SPLICE it was altered as shown in Fig. 4.10. The additional elements are the interface elements required by SPLICE where logical and electrical circuit elements are connected to one another. The SPLICE input file which describes this circuit is shown in Fig. 4.11.

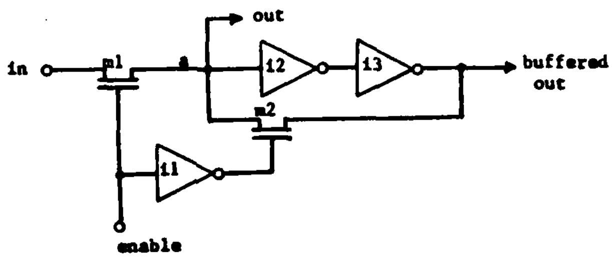


Fig. 4.9 Static Register

For this example positive logic is assumed, v0= 0v and v1= 5v. When enable is 5v, m1 is on and m2 is off. The voltage value at the input is passed through m1 to a . The value at e is a buffered replica of the input. When enable is 0v, m1 is off and m2 is on. The value at e is fed back through m2 to node a .

The voltage levels at node a are determined as follows. Va(0) is simply zero when m1 is on. In this case input= 0v and is passed through saturated m1. When m2 is on Va(0) is approximately equal to the v0 value from the logic-to-voltage converter between i3 and m2, or 0.5v. Va(1) with

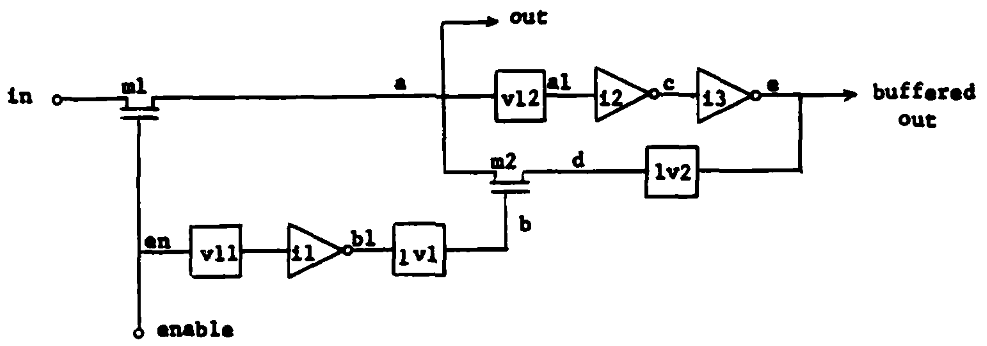


Fig. 4.10 Static Register (with interface elements)

m1 or m2 on is approximately equal to 5 volts minus the threshold voltage of m1 and m2. For this example Va(1) = 5 - 1.0 or 4v. The results from SPLICE were:

```
m1 on -- Va(0)= 0.0v  Va(1)=3.89v
m2 on -- Va(0)= 0.5v  Va(1)=3.86v
```

The parallel plot (timing diagram) output from this file is shown in Fig. 4.12. The time and pstep statements in this file requested an anlaysis from 0 to 400 ns in 1ns increments, plotting every 10th point. This file required 6.6 seconds of CPU time on the VAX/UNIX. With the pstep 10 statement deleted and the time statement changed to 10ns 400 ns the file was requesting an analysis from 0 to 400ns in 10ns increments plotting each point. With these changes the file required 2.5 seconds of CPU time. The outputs were approximately the same. The voltage waveform at node a, with 10ns stepsize, was a little more ragged but the buffered output at node e was the same clean waveform.

```
splice
static register
;
;models
;
model input  tsrc      : 5 0 0 400n 0n 200n 210n 400n
model enable tsrc      : 5 0 0 400n 0n 100n 101n 149n 150n $
                                240n 241n 350n 351n 400n
model inv    inverter : tr=10ns tf=10ns
model mos1   nmos      : w=5u l=1u vto=1.0
model vtl    vtl       : v0=2.0 v1=2.5
model ltv    ltv       : v0=.5 v1=5.0
model gc     gcapr     : 1pf
;
;elements
;
m1   in   en    a   mos1
m2   d    b     a   mos1
i1   b1   en1       inv
i2   c    a1        inv
i3   e    c         inv
in   in             input
en   en             enable
c1   in             gc
c2   en             gc
c3   a              gc
c4   b              gc
c5   d              gc
vl1  en1  en        vtl
vl2  a1   a         vtl
lv1  b    b1        ltv
lv2  d    e         ltv
;
;analysis requests
;
time 1n  400n
pstep 10
wplot in en a e
print in en a e
stats
;
go
;
end
```

Fig. 4.11 SPLICE Input File - Static Register

For very large LSI type circuits it would be too cumbersome to include discrete interface elements between electrical and logical elements. If the same logic levels are used in the entire circuit 'vlevels' and 'llevels' can be used to implement the necessary conversions. There were some problems encountered when I tried to use these options, however.

When the static register circuit in Fig 4.9 was run with the 'vlevels' and 'llevels' options selected instead of discrete interface elements, an error message was generated suggesting zero-delay feedback in the logic circuit. Nothing in the circuit description could be found which would cause this error. I found the probable cause of the problem when I added a logic inverter to the output node of the discrete NMOS inverter circuit from the test circuit in Fig 4.1, and specified implicit conversion with 'vlevels' and 'llevels'. The implied conversion was producing an output change before the input changed. This would explain the zero delay error. This suggests a problem in the source code of SPLICE. Until it is corrected, SPLICE seems very cumbersome for use in LSI circuit simulation.

Since SPLICE was run on three different computers, some comparisons can be made. Some are subjective and depend on personal preferences, like user-friendliness and editor utility.

One very noticeable difference was the computation time required by the differenct machines in the processing of SPLICE files. In almost every circuit run the VAX/VMS was faster than the VAX/UNIX and the IBM was much faster than the other two. For example the NMOS inverter circuit above took 1.017 seconds to run on the VAX/UNIX, .890 second on the VAX/VMS, and .670 second on the IBM. Similarly for the T flip-flop circuit, which was only run on two machines. On the VAX/UNIX it took 2.850 seconds and on the IBM it took only .636 second.

```
  **** splice version 1A.3 ****
static register;

legend:
+ <=> in       :elect     :
> <=> en       :elect     :
< <=> a        :elect     :
& <=> e        :elect     :
 scale: minimum= 0.000e+00 maximum= 5.000e+00
   time
0.000e+00.                  +.                  >.   <                .         &         .
1.000e-08.                  +.                  >.           <        .         &         .
2.000e-08.                  +.                  >.             <      .         &         .
3.000e-08.                  +.                  >.             <      .                  &.
4.000e-08.                  +.                  >.              <     .                  &.
5.000e-08.                  +.                  >.              <     .                  &.
6.000e-08.                  +.                  >.              <     .                  &.
7.000e-08.                  +.                  >.               <    .                  &.
8.000e-08.                  +.                  >.               <    .                  &.
9.000e-08.                  +.                  >.               <    .                  &.
1.000e-07.                  +.                  >.               <    .                  &.
1.100e-07.                  +.                  >.               <    .                  &.
1.200e-07.                  +.>                  .               <    .                  &.
1.300e-07.                  +.>                  .               <    .                  &.
1.400e-07.                  +.>                  .               <    .                  &.
1.500e-07.                  +.                  >.               <    .                  &.
1.600e-07.                  +.                  >.               <    .                  &.
1.700e-07.                  +.                  >.               <    .                  &.
1.800e-07.                  +.                  >.               <    .                  &.
1.900e-07.                  +.                  >.               <    .                  &.
2.000e-07.                  +.                  >.               <    .                  &.
2.100e-07.   +               .                  >.            <       .         &         .
2.200e-07.+                  .                  >.<                   .         &         .
2.300e-07.+                  .                  >.<                   .&                  .
2.400e-07.+                  .                  >.<                   .&                  .
2.500e-07.+                  .>                  .<                   .&                  .
2.600e-07.+                  .>                  . <                  .&                  .
2.700e-07.+                  .>                  . <                  .&                  .
2.800e-07.+                  .>                  . <                  .&                  .
2.900e-07.+                  .>                  . <                  .&                  .
3.000e-07.+                  .>                  . <                  .&                  .
3.100e-07.+                  .>                  . <                  .&                  .
3.200e-07.+                  .>                  . <                  .&                  .
3.300e-07.+                  .>                  . <                  .&                  .
3.400e-07.+                  .>                  . <                  .&                  .
3.500e-07.+                  .>                  . <                  .&                  .
3.600e-07.+                  .                  >.<                   .&                  .
3.700e-07.+                  .                  >.<                   .&                  .
3.800e-07.+                  .                  >.<                   .&                  .
3.900e-07.+                  .                  >.<                   .&                  .
```

Fig. 4.12 Parallel Plot - Static Register

CHAPTER V

Results and Conclusions

5.1 Results

The main objectives of the project were to modify SPLICE version 1A.2 from VAX/VMS format to the IBM 4341 and to simulate test circuits with SPLICE and evaluate its performance as a simulator. Both objectives were accomplished.

The VAX/VMS version of SPLICE was copied on to the IBM 4341 disk. Then it was debugged to clear the errors caused by differences in the Fortran used by the two systems. Errors caused by other system differences were also eliminated. Procedures were developed to analyze test circuits on the IBM. They are described in Appendix A. Procedures to run SPLICE on the VAX/UNIX are also included. They are in Appendix B.

Test circuits were analyzed using SPLICE. Analog, logical and mixed analog and logical circuits were used.

5.2 Conclusions

The following conclusions can be made about SPLICE.

1. SPLICE is not a general purpose mixed-mode circuit simulator. It has a limited number of analog element types. It does not include bipolar tran-

sistors, diodes, inductors, AC sources, and others. SPLICE was designed to simulate MOS LSI circuits.

2. SPLICE is limited in its ability to simulate LSI circuits. The implicit interface between logic and electrical elements is essential if SPLICE is to effectively be used to simulate large circuits.
3. SPLICE performs logic analysis with delays and analog analysis with accuracy comparable to SPICE. With discrete interface elements it analyzes mixed-mode circuits consisting of logical gates and MOSFET switching circuits.
4. Run times for all test circuits were shorter on the VAX/VMS than on the VAX/UNIX. Run times on the IBM 4341 were shorter than on either VAX machine. This could be an important factor on large circuits.

To summarize, the objectives of the project were to implement SPLICE on the IBM 4341 and to investigate SPLICE as a mixed-mode circuit simulator. SPLICE is installed and running on the IBM 4341 system. SPLICE has some limitations as a mixed-mode simulator. It is limited in available analog elements and is tedious to use for LSI circuits. An error was found in the SPLICE utilities for implicit interface between analog and logic circuits.

References

1. Jensen, Randall W. and McNamee, Lawrence P., Handbook of Circuit Analysis Languages and Techniques, Prentice-Hall inc., Englewood Cliffs, New Jersey, 1976.

2. Horowitz, Paul and Hill, Winfield, The Art of Electronics, Cambridge University Press, Cambridge, 1980.

3. SPLICE Version 1A.2 User's Guide, Electronics Research Laboratory, University of California, Berkeley, 1980.

4. SPLICE Version 1.6 User's Guide, Electronics Research Laboratory, Univeristy of California, Berkeley, 1983.

5. Newton, A. Richard, Techniques for the Simulation of Large-Scale Integrated Circuits, IEEE Transactions on Circuits and Systems, Volume CAS-26, No. 9, New York, 1979, pp 741-749.

6. Mavor, J., Jack M.A., and Denyor, P.B., Introduction to MOS LSI Design, Addison-Wesley Publishing Company, London, 1983.

7. Elmary, Muhammed I., Digital MOS Integrated Circuits, Institutate of Electrical and Electronics Engineers Press, New York, 1981.

8. Barna, Arpad and Porat, Dan I., Integrated Circuits in Digital Electronics, Wiley, New York, 1973.

9. Mead, Carver and Conway, Lynn, Introduction to VLSI Systems, Addison-Wesley, Reading, Massachusetts, 1980.

APPENDIX A:

Running SPLICE on the IBM 4341

It is assumed that users know how to log on and off the system and how to use the XEDIT utility to create and edit files.

1. Log on using normal 4341 login procedures.
2. After log on is complete, it is necessary to increase the virtual storage. To do this type:

    ```
    def stor 2m <enter>
    ```

3. Now it is necessary to re-enter CMS. Type:

    ```
    ipl cms <enter>
    ```

4. Press <enter> again, and the following message will appear:

    ```
    DMSINS100W CMSSEG SYSTEM NAME 'CMSSEG' NOT
    AVAILABLE
    ```

5. Press <enter> and wait for a CMS ready message.
6. Now a SPLICE input file is needed. Use XEDIT to create a file with filename = test, filetype = spl, and filemode = a.

    Fig A.1 is an example of a SPLICE input file. This file describes the logic circuit shown in Fig A.2. It requests a timing diagram plot and a listing of the inputs, a and b, and the output, out.

```
splice
exclusive-or
;
;models
;
model ain lsrc : 0 1 0 40u 0u 19u 20u 39u 40u
model bin lsrc : 0 1 0 20u 0u  9u 10u 19u 20u
model and and  : tr=5n tf=5n
model inv inv  : tr=5n tf=5n
model or  or   : tr=5n tf=5n
;
;elements
;
ain   5             ain
bin   6             bin
b1    a    5        buf
b2    b    6        buf
i1    1    a        inv
i2    2    b        inv
a1    3    2    a   and
a2    4    1    b   and
o1    out  3    4   or
b3    7    out      buf
;
;analysis requests
;
time 1u 40u
wplot  a   b   out
print  a   b   out
;
go
;
end
```

Figure A.1 Example of a SPLICE input file

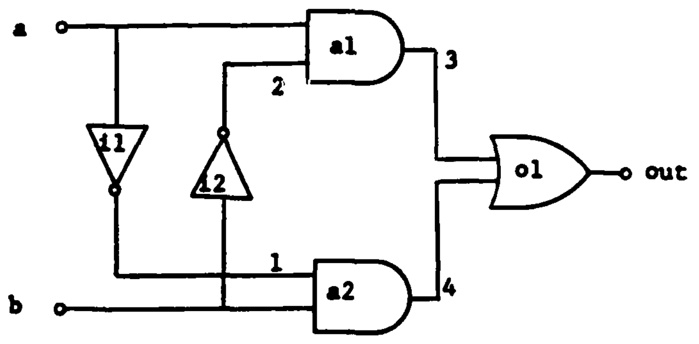


Figure A.2 Example Circuit (exclusive-or)

7. After an input file is created the following steps are used to process it.

   Type:

   blt <enter>

An flist listing of the file BLT ERROR A will appear. This file contains a listing of the input file, any errors detected by BLT, and a node map. If there are errors detected by BLT, determine which errors occurred and use XEDIT to make the necessary corrections to TEST SPL A. After the corrections have been made, type:

```
clear <enter>
```

and then type:

```
blt <enter>
```

Repeat this cycle until there are no errors detected by BLT. When BLT runs error-free type:

```
splice <enter>
```

Either a CMS Ready message or an flist listing of SPLICE ERROR A will now appear. A CMS Ready message looks like this:

```
R; T=0.08/0.16 12:44:30
```

If the error file appears examine it to determine which errors have been detected by SPLICE and make the necessary corrections to TEST SPL A. After making a correction it will be necessary to run BLT again as above and then run SPLICE again. Repeat until 'splice <enter>' produces a CMS Ready message.

When a CMS Ready message appears, type:

```
splot <enter>
```

An flist display of the output file, SPLOT OUT A, will appear on the screen when SPLOT has finished. The output file should be examined and then may be printed. To print a hard copy of SPLOT OUT A, type:

```
print splot out a <enter>
```

APPENDIX B

Running SPLICE on the VAX/UNIX

These are the instructions for running SPLICE on the VAX/UNIX. It is assumed that users know how to log on and off and how to use the editor to create and edit files.

1. Log on using normal VAX/UNIX login procedures.
2. After logged on, use the editor to create a SPLICE input file. An example file can be found in Appendix A. The file can be named any valid file-name.
3. After the input file is created, it is processed as follows:

   Type:

   ```
   blt -i filename1 -o filename2
   ```

   Here filename1 is the name of the SPLICE input file that was created in step 2 above. The same format is used to run SPLICE and SPLOT. Assume the SPLICE input file created above is named 'test', the output file from BLT is named 't.b', the output file from SPLICE is named 't.s', and the output file from SPLOT is named 'testout', then type:

   ```
   blt -i test -o t.b
   ```

   If BLT detects any errors it will print them on the screen.

If any are found use the editor to make the necessary corrections to 'test', then run BLT again. Repeat this cycle until no errors are found. When there are no errors the prompt , % , will appear. When BLT runs error-free and the prompt , % , appears, type:

```
splice -i t.b -o t.s
```

As with BLT, errors will be printed on the screen. If errors are found 'test' must be corrected and processed by BLT again and then by SPLICE again. Repeat this cycle until SPLICE runs error-free. When SPLICE runs error-free and the prompt, % , appears type:

```
splot -i t.s -o testout
```

Again there may be errors detected, this time by SPLOT. The appropriate corrections need to be made to 'test' and the complete cycle repeated until BLT, SPLICE, and SPLOT all run error-free. Even if there are errors detected by SPLOT there may be an output file generated. If it is, it can be examined to help in determining what corrections need to be made to 'test'. If no errors are detected by SPLOT the prompt , % , will appear on the screen. The output file can be examined by typing:

```
cat testout
```

The output file, testout, can be sent to the printer by typing:

```
lpr testout
```